



한국어와 NLTK, Gensim의 만남

(부제: 영화 리뷰를 컴퓨터가 이해할 수 있는 형식으로 표현해서 센터멘트 분석하기)

2015-08-29

Lucy Park (박은정)

개요

KoNLPy를 쓰면 이제 파이썬으로도 형태소 분석을 하거나 워드 클라우드를 그릴 수 있다던데, 이거 말고 또 할 수 있는건 없나요? 있습니다!

파이썬의 유명한 자연어 처리 패키지 NLTK를 활용하면 문서 안의 내용을 빠르게 탐색하거나 요약할 수 있고, 토픽 모델링을 지원하는 패키지 Gensim을 사용하면 여러 문장이나 문서에 내재되어 있는 규칙, 또는 토픽들을 찾아낼 수 있다.

이 발표에서는 먼저 bag-of-words, document embedding 등 컴퓨터가 문서를 잘 분석 할 수 있게 하는 다양한 텍스트의 표현 방법에 대해 살펴본 후, KoNLPy, NLTK, Gensim 등의 라이브러리를 실제 한국어 문서들에 적용해본다.

- KoNLPy: <http://konlpy.org>
- NLTK: <http://nltk.org>
- Gensim: <http://radimrehurek.com/gensim>



박은정

(a.k.a. lucypark,
echojuliett, e9t)

개발하는 데이터 분석가.

- 서울대학교 데이터마이닝 센터 박사과정
- 대한민국 정치의 모든 것 만드는 팀포퐁 멤버
- CAVEAT: KoNLPy 메인테이너
- Just another yak shaver...

11:49 <@sanxiyn> 또다시 yak shaving의 신비한 세계
11:51 <@sanxiyn> yak shaving이 뭔지 다 아시죠?
11:51 <디토군> 방금 찾아보고 왔음
11:51 <@mana> (조용히 설명을 기대중)
11:51 <@sanxiyn> 나무를 베려고 하는데
11:52 <@sanxiyn> 도끼질을 하다가
11:52 <@sanxiyn> 도끼가 더 잘 들면 나무를 쉽게 벨텐데 해서
11:52 <@sanxiyn> 도끼 날을 세우다가
11:52 <@sanxiyn> 도끼 가는 돌이 더 좋으면 도끼 날을 더 빨리 세울텐데 해서
11:52 <@sanxiyn> 좋은 숫돌이 있는 곳을 수소문해 보니
11:52 <@mana> ...
11:52 <&홍민희> 그거 전형적인 제 행동이네요
11:52 <@sanxiyn> 저 멀리 어디에 세계 최고의 숫돌이 난다고
11:52 <@sanxiyn> 거기까지 악크를 타고 가려다가
11:52 <@mana> 항상하던 짓이라서 타이핑을 할 수 없었습니다
11:52 <@sanxiyn> 악크 털을 깎아서...
11:52 <@sanxiyn> etc.

발표에 관하여

이 발표가 다루는 내용

- 단어/문서를 컴퓨터가 이해할 수 있게 하는 표현 방법
- KoNLPy, NLTK, Gensim의 간단한(?) 사용법
- 이 발표를 듣고 나면 바로 센터멘트 분석을 할 수 있습니다!

이 발표가 다루지 않는 내용

- 토픽 모델링
 - Gensim에 포함되어 있지만, 오늘은 일관된 흐름을 위해 다루지 않습니다.
- 아쉽지만, 알고리즘에 대한 세세한 내용도 배제
 - 대신 주석으로 달린 참고문헌들을 봐주세요.
- 패키지들의 세부사항도 다루지 않습니다.

발표를 들으면서 하면 좋은 생각

- 어디에 써먹을 수 있을까?

기타사항

- 코드는 파이썬 3 기준

1997년

지금으로부터 약 20년 전 ...



IBM 딥블루가 체스 세계 챔피온 가리 카스파로프를 이기다.



체스는 몇 가지 정형화된 규칙으로도 충분히 정의할 수 있기 때문에 그것을 컴퓨터에게 가르치는 것은 제법 쉽다. 하지만, 사람이 물체를 인식하고 말하는 것을 정형화된 규칙으로 정의하기는 쉽지 않기 때문에 컴퓨터에게 그런 능력을 주는 것도 쉽지 않다.

컴퓨터가 더 똑똑해지기 위해서는
정형화된 규칙이 없는 지식을 컴퓨터에게 어떻게 전달
할 것인지 알아야 한다.

-- Bengio et al., **Deep Learning** 1장 본문 의역, Book in preparation for MIT Press, 2015.

텍스트를 "표현"하는 다양한 방법

Representations of text

어떻게 하면 구조가 없는 데이터인 텍스트의 의미를 컴퓨터가 잘 이해할 수 있을까?

이진 파일



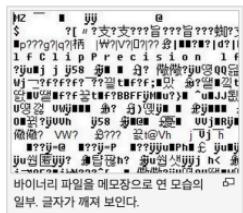
이 문서의 내용은 **출처**가 분명하지 않습니다.

지금 바로 이 [문서를 편집](#)하여, 참고하신 문헌이나 **신뢰할 수 있는 출처**를 각주 등으로 표기해 주세요.

검증되지 않은 내용은 삭제될 수도 있습니다. 내용에 대한 의견은 [토론 문서](#)에서 나누어 주세요. (2012년 8월 8일에 문서의 출처가 요청되었습니다.)

이진 파일 또는 바이너리 파일(binary file)은 **컴퓨터 파일**로 컴퓨터 저장과 처리 목적을 위해 **이진 형식**으로 인코딩된 데이터를 포함한다. 이를테면, **포맷 텍스트**를 포함하는 **컴퓨터 문서 파일**을 수 있다. 많은 이진 파일 형식은 문자열로 해석될 수 있는 부분을 포함하고 있다. 포맷 정보가 없는 문자열 데이터만 포함하는 이진 파일은 **완전한 텍스트 파일**이라고 한다. 텍스트로만 이루어진 파일은 보통 이진 파일과 구분짓는데 이진 파일은 완전한 텍스트 이상의 무언가를 더 포함하고 있기 때문이다.

이진 파일은 텍스트 모드로 열면 다음 그림과 같이 글자가 깨진다.



```
00000000 0000 0001 0001 1010 0018 0001 0004 0128  
00000012 0000 0016 0000 0028 0000 0010 0000 0020  
00000020 0000 0001 0004 0000 0000 0000 0000 0000  
00000030 0000 0000 0000 0018 0000 0000 0000 0000 0000  
00000040 0004 8384 0084 c7c8 00c9 4748 0048 e9e9  
00000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc  
00000060 00fc 1819 0019 9598 0098 9d98 00d8 5857  
00000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888  
00000080 8888 8888 8888 8888 288e be88 8888 8888  
00000090 3b63 5788 8888 8888 7667 7786 0028 8888  
000000a0 d61f 7abd 8818 8888 467c 585f 8814 8188  
000000b0 b5b6 e8f7 88aa 8388 8130 8873 88d0 e988  
000000c0 5a18 8888 8888 c988 b328 8888 9586  
000000d0 9494 5862 5864 7e80 3780 1bb4 5ab4 3ec0  
000000e0 3d68 dc68 5ccb 8888 8888 8888 8888 8888  
000000f0 8888 8888 8888 8888 8888 8888 8888 8888  
00000100 0000 0000 0000 0000 0000 0000 0000 0000 0000  
*  
00000120 0000 0000 0000 0000 0000 0000 0000 0000 0000  
0000012e
```

위키백과 아이콘 W은 **텍스 덤프**로 318 바이트를 차지 한다. 첫 열은 각 행의 시작 주소를 나타내고, *는 반복되는 데이터로 생략된 주소를 나타낸다.

좌측은 사람이 읽는 텍스트, 우측은 컴퓨터가 읽는 바이너리.
하지만 바이너리에는 "의미 정보"가 담겨있지 않다.

바이너리 파일을 메모장으로 연 모습의 Ⓜ

일부 글자가 깨져 보인다.

텍스트의 의미를 이해한다
≈ 유사한 의미를 가진 텍스트끼리 묶을 수 있다

{단어, 문서} ⊂ 텍스트

먼저, 단어를 표현하는 가장 쉬운 방법은,
이진(binary) 표현법

$$w^{\text{가}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{\text{가가(假家)}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{\text{가가대소}} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots w^{\text{활자}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

- w^i 은 i 번 단어에 대한 벡터 표현
- 각 벡터의 크기는 모든 단어의 set인 전체 사전의 크기 $|V|$ 와 같다
- 뉴럴넷 쪽 학자들은 이렇게 표현한 벡터를 보통 one-hot vector라고 한다
- 어떤 사람들은 이런 방식의 인코딩을 1-of-V 코딩이라고도 한다

하지만 이진 표현법을 사용하면 단어 간 유사도를 정의할 수 없다.

- ex: "호텔"과 "모텔", "호텔"과 "고양이" 사이의 거리가 전부 0으로 같다.

$$(w_{\text{호텔}})^T w_{\text{모텔}} = (w_{\text{호텔}})^T w_{\text{고양이}} = 0$$

같은 철학으로 문서를 표현한 것이,
bag of words(BOW)

- 단어가 문서에 존재한다/안한다 (이 발표에서는 "term existance"라고 부름)

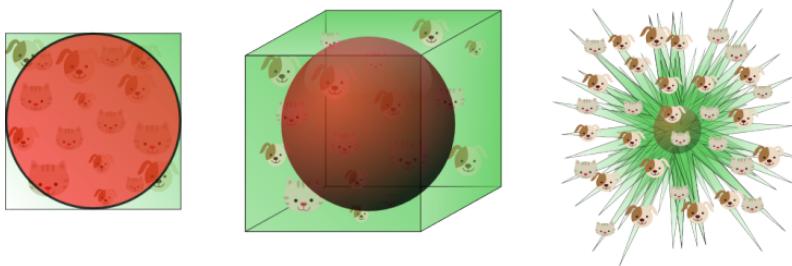
$$d_{binary}^1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- 단어가 문서에 n번 존재한다 (a.k.a. term frequency (TF))

$$d_{tf}^1 = \begin{bmatrix} 3 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- 단어에 중요도를 할당하고 문서에 등장한 단어 중요도의 가중합을 구한다
 - ex: term frequency-inverse document frequency (TF-IDF)

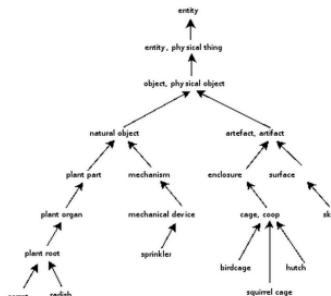
그런데 공간의 차원이 너무 커서 문서 간 유사도 지표의 효과가 떨어진다.*



- 공간의 차원이 커질수록 데이터의 대부분은 공간의 꼭지에 분포하게 된다.
- 문서 간 거리의 편차가 매우 적어지게 됨.
- 문서 간 유사도가 "또이또이".

"그럼 워드넷 같은 텍소노미를 활용하는건 어때요?"

워드넷(WordNet)은 X is-a Y와 같은 상위어(hypernym) 관계 등을 정의하는 방향성 비순환 그래프(DAG; directed acyclic graph)입니다. (예: 사과는 과일이다.)



```
from nltk.corpus import wordnet as wn

wn.synsets('computer') # "computer"의 유사어를 반환
#=> [Synset('computer.n.01'), Synset('calculator.n.01')]
```

참고: 한국어 어휘의 미망(워드넷)은 **부산대** 등에 의해 개발/공개됨

하지만 워드넷은 다음의 단점이 있습니다:

1. 모든 용어를 포함하려고는 하지만, 전문 도메인 용어들은 많이 빠져 있다.

```
from nltk.corpus import wordnet as wn
wn.synsets('nginx')
#=> []
```

2. 각종 신조어에 대한 정보를 확보하기 위해서는 꾸준히 유지보수를 해줘야 한다.

```
wn.synsets('yolo') # 흔: you only live once
#=> []
```

3. 게다가 NLTK로는 한국어 지원이 안됨... (중요!)

```
wn.synsets('컴퓨터', lang='kor')
#=> WordNetError: Language is not supported.
```

아..안되겠다. 다른 방법 없나?

친구를 보면 그 사람을 안다.

類類相從

단어의 주변을 보면 그 단어를 안다.

You shall know a word by the company it keeps.

-- 언어학자 J.R. Firth (1957)

예시 1: 빈 칸 안에 맞는 말?

- 새로운 디자인의 선이 하다.
- 야, 부엌 좀 히 하자.
- 깜박하고 책상 위를 하게 치우지 않았다.

긁으면 정답: []

예시 2: "매나니"라는 단어를 아시나요?

なに?

- 그 녀석 무슨 배짱인지 **매나니**로 와서 일을 하겠다고 한다.
- 삽이라도 있어야 땅을 파지 **매나니**로야 어떻게 하겠나?
- SO만 있으면 코딩은 **매나니**로도 할 수 있다고 한다.

매나니

♦ 단어장 저장

관련 어휘



명사

1. 무슨 일을 할 때 아무 도구도 가지지 아니하고 맨손뿐인 것.
 - 삼이라도 있어야 땅을 파지 매나니로야 어떻게 하겠나?
2. 반찬 없는 맨밥.

부사

[북한어] 아무것도 하지 아니하고. 또는 아무 생각도 없이 그저 명하니.

- 매나니 기다리다
- 매나니 근심만 하지 말고 대책을 세워야지. 출처 : 조선말 대사전(1992)

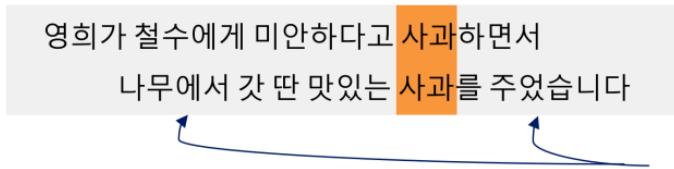
관련 규범 해설

‘매나니’의 의미로 ‘맨털터리’를 쓰는 경우가 있으나 ‘매나니’마 표준어로 삼는다.

관련조항 : 표준어 규정 3장 4절 25항

실존하는 단어;;

다시 말해, 단어의 의미는 해당 단어의 **문맥(context)**이 담고 있다.



주변부 단어들이 “사과”的 문맥적 특성을 나타냄

- 문맥(context) := 정해진 구간(window) 또는 문장/문서 내의 단어들
 - 보통 영어의 구간은 좌우 1-8개 단어, 총 3-17개 단어의 문맥을 본다*
- 두 단어의 문맥이 비슷하면 "의미적"으로 유사한 단어
 - ex: "PyCon"은 "R"보다 "Python"에 가깝다?

"Co-occurrence"

두 단어가 정해진 구간 내에서 동시에 등장함

Co-occurrence를 정의하는 두 가지 방법

```
documents = [["나는", "파이썬", "이", "좋다"],  
             ["나는", "R", "이", "좋다"]]
```

1. **Term-document matrix** ($X_{td} \in \mathbb{R}^{|V| \times M}$): 한 문서에 같이 등장하면 비슷한 단어
 - Computation이 문서의 개수 M 에 비례(!)

```
x_td = [[1, 1], # 나는  
         [1, 0], # 파이썬  
         [0, 1], # R  
         [1, 1], # 이  
         [1, 1]] # 좋다
```

1. **Term-term matrix** ($X_{tt} \in \mathbb{R}^{|V| \times |V|}$): 단어가 문맥 내에 같이 존재하면 비슷한 단어
 - 문맥의 길이가 짧을수록 syntactic, 길수록 semantic 정보를 포함
 - 앞뒤로 단어를 두 개씩 보는 경우 (문맥의 길이==5):

```
x_tt = [[0, 1, 1, 2, 0], # 나는  
         [1, 0, 0, 1, 1], # 파이썬  
         [1, 0, 0, 1, 1], # R  
         [2, 1, 1, 0, 2], # 이  
         [0, 1, 1, 2, 0]] # 좋다
```

Co-occurrence matrix를 있는 그대로 이용해도 단어 간 유사도를 구할 수는 있다.

```
import math

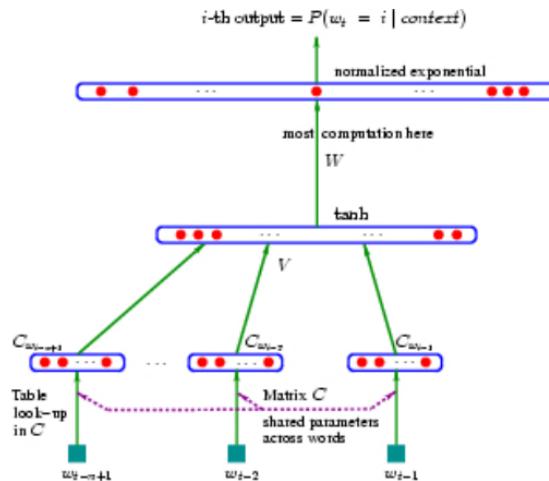
def dot_product(v1, v2):
    return sum(map(lambda x: x[0] * x[1], zip(v1, v2)))

def cosine_measure(v1, v2):
    prod = dot_product(v1, v2)
    len1 = math.sqrt(dot_product(v1, v1))
    len2 = math.sqrt(dot_product(v2, v2))
    return prod / (len1 * len2)
```

- 하지만:
 - 값들이 너무 skewed 되어 있고 (즉, 빈도 높은 단어와 낮은 단어의 격차가 큼)
 - 정보성 낮은 단어 때문에 discriminative하지 않음 (ex: list('은는이가'))
- 유사도 정보가 담긴 단어 벡터를 구하는 더 나은 방법은?

Neural embeddings

- 아이디어: 문맥에 있는 단어를 예측하라!
 - 언어 모델(language model) 활용
 - ex: ["나는", "파이썬", "이", "좋다"] 다음에 뭐가 나올까? (ex: "!", ".", "?")
- 그리고, 학습할 때 뉴럴넷을 쓰자.

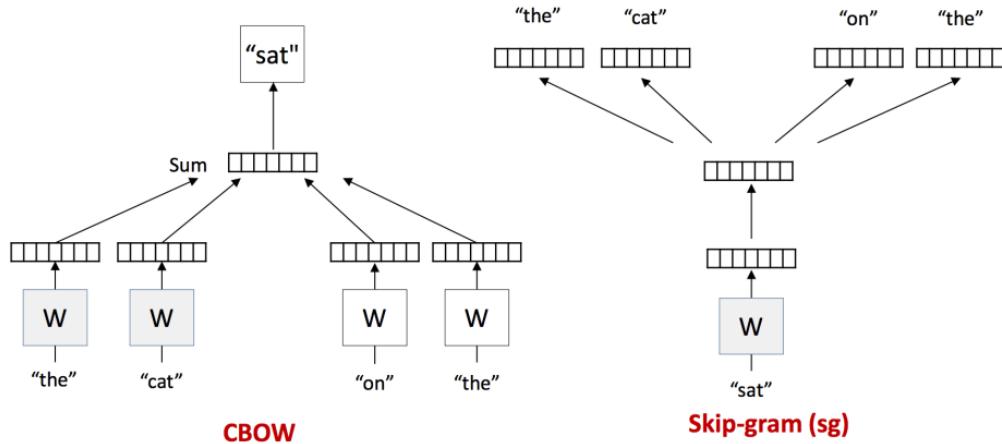


Neural network language model (NNLM)*

* Bengio, Ducharme, Vincent, A Neural Probabilistic Language Model, {NIPS, 2000; JMLR, 2003}.

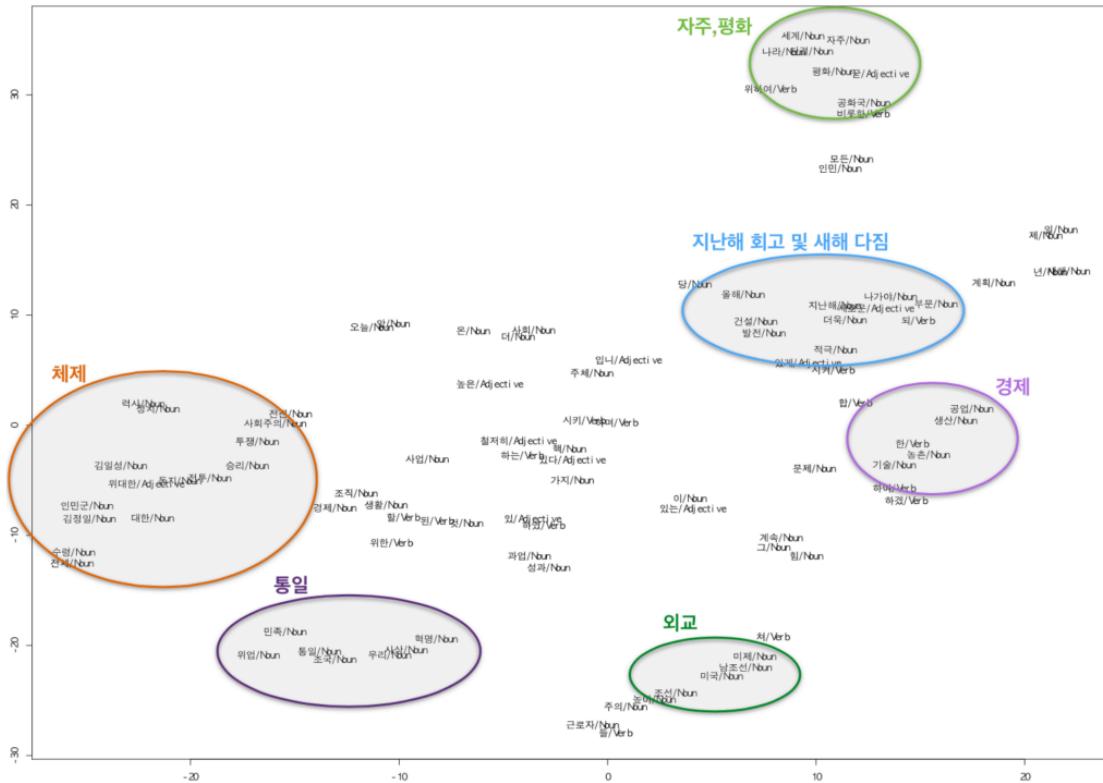
word2vec *

- 요즘 핫한 T. Mikolov의 word2vec도 neural embedding
- 각종 계산 트릭을 적용해서 컴퓨테이션이 빨라짐! (n 일 $\rightarrow m$ 시간)



* Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionalities. In Proceedings of NIPS, 2013.

예시: 북한 신년사에 word2vec을 적용 *

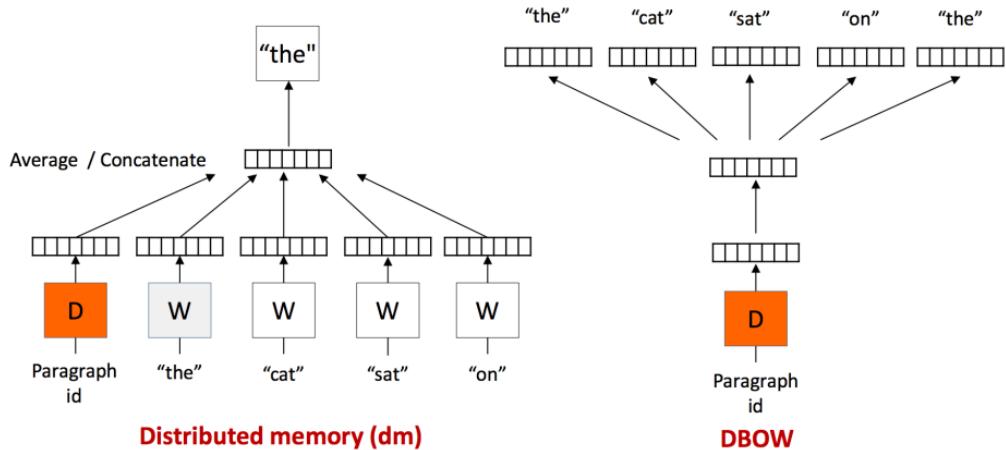


* 박종희, 박은정, 조동준, 북한 신년사(1946-2015)에 대한 자동화된 텍스트 분석, *한국정치학회보* 제49집 제2호, 2015.

그럼 문서에 대한 neural embedding은?

doc2vec *

- 아이디어: 문서(또는 문단) 벡터를 마치 단어인 양 학습시키자!
- 장점
 - 차원이 $|V|$ 에 비해 훨씬 적어진다.
 - 단어 벡터와 같은 공간에 문서를 전사할 수 있다.



* Le, Mikolov, *Distributed Representations of Sentences and Documents*, ICML, 2014. (원문에서는 doc2vec 대신 **paragraph vector**라는 이름을 사용함)

예시: Term frequency -> 문서 벡터의 크기가 단어의 수 $|V|$ 와 같음. 원소들은 양의 정수.

```
documents = [  
    ['왕자', '가', '공주', '를', '좋아한다'],  
    ['공주', '가', '왕자', '를', '좋아한다'],  
    ['시녀', '가', '왕자', '를', '싫어 한다'],  
    ['공주', '가', '시녀', '를', '싫어 한다'],  
    ['시녀', '가', '왕자', '를', '독살한다'],  
    ['시녀', '가', '공주', '를', '독살한다']  
]  
words = set(sum(documents, []))  
print(words)  
# => {'가', '공주', '독살한다', '를', '시녀', '싫어 한다', '왕자', '좋아한다'}
```

```
def term_frequency(document):  
    # do something  
    return vector  
  
vectors = [term_frequency(doc) for doc in documents]  
print(vectors)  
# => {  
#     'S1': [1, 1, 0, 1, 0, 0, 1, 1],  
#     'S2': [1, 1, 0, 1, 0, 0, 1, 1],  
#     'S3': [1, 0, 0, 1, 1, 1, 1, 0],  
#     'S4': [1, 1, 0, 1, 1, 1, 0, 0],  
#     'S5': [1, 0, 1, 1, 1, 0, 1, 0],  
#     'S6': [1, 1, 1, 1, 1, 0, 0, 0]  
# }
```

예시: doc2vec -> 문서 벡터의 크기가 단어의 수 $|V|$ 보다 작음. 원소들은 실수.

```
documents = [
    ['왕자', '가', '공주', '를', '좋아한다'],
    ['공주', '가', '왕자', '를', '좋아한다'],
    ['시녀', '가', '왕자', '를', '싫어 한다'],
    ['공주', '가', '시녀', '를', '싫어 한다'],
    ['시녀', '가', '왕자', '를', '독살한다'],
    ['시녀', '가', '공주', '를', '독살한다']
]
words = set(sum(documents, []))
print(words)
# => {'가', '공주', '독살한다', '를', '시녀', '싫어 한다', '왕자', '좋아한다'}
```

```
def doc2vec(document):
    # do something else (뒤에서 Gensim이 해줄 것!)
    return vector
```

```
vectors = [doc2vec(doc) for doc in documents]
print(vectors)
# => {
#     'S1': [-0.01599941, 0.02135301, 0.0927715],
#     'S2': [0.02333261, 0.00211833, 0.00254255],
#     'S3': [-0.00117272, -0.02043504, 0.00593186],
#     'S4': [0.0089237, -0.00397806, -0.13199195],
#     'S5': [0.02555059, 0.01561624, 0.03603476],
#     'S6': [-0.02114407, -0.01552016, 0.01289922]
# }
```

요약

- 단어의 문맥을 이용해서 단어를 표현한 수 있다.
- 텍스트의 의미를 벡터로 표현하는 몇 가지 방법:

	Sparse, long vectors	Dense, short vectors
단어	1-hot-vectors	word2vec
문서	bag-of-words (term existance, TF, TF-IDF 등)	doc2vec

이제 이론을 익혔으니 실전으로 가보자.

한국어 영화 리뷰 센티멘트 분석

Sentiment analysis for Korean movie reviews

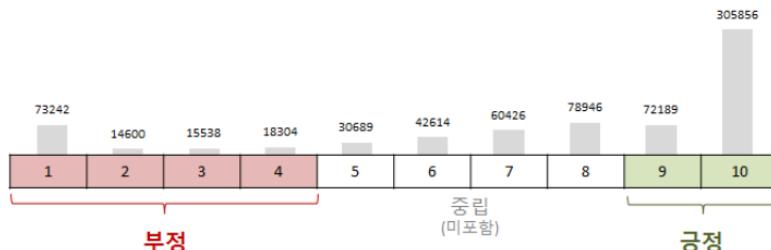
잠깐, 데이터는 어디서 구하지?

한국어 영화 리뷰 토이 데이터 공개

토이 데이터가 많아야 학문이 발전한다!

Naver sentiment movie corpus v1.0

- Maas et al., 2011 데이터셋 참고
- 데이터 출처: 네이버
- 영화 당 100개의 140자평(이하 '리뷰')을 초과하지 않음
- 총 20만 개 리뷰 (수집된 64만개 중 샘플링)
 - ratings_train.txt: 15만, ratings_test.txt: 5만
 - 긍정/부정 리뷰의 비율을 동일하게 샘플링 (i.e., random guess yields 50% accuracy)
 - 중립 리뷰는 포함하지 않음



- 크기: 19MB
- URL: <http://github.com/e9t/nsmc/>

This repository Search

Pull requests Issues Gist

e9t / nsmc

Unwatch 1 Star 0 Fork 0

Naver sentiment movie corpus v1.0 <http://lucypark.kr/slides/2015-pyconkr/#40> — Edit

3 commits 1 branch 0 releases 1 contributor

Branch: master nsmc / +

Modify headers

e9t authored 5 hours ago latest commit eafdd77e31

ratings.txt Initial commit 11 hours ago

ratings_test.txt Modify headers 5 hours ago

ratings_train.txt Modify headers 5 hours ago

Add a README

Help people interested in this repository understand your project by adding a README.

SSH clone URL

git@github.com:e9t/nsmc

You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).

Clone in Desktop Download ZIP

README는 coming soon

```
$ cat ratings_train.txt | head -n 25
```

id	document	label
----	----------	-------

9976970	아 더빙.. 진짜 짜증나네요 목소리	0
3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
10265843	너무재미없었다그래서보는것을추천한다	0
9045019	교도소 이야기구먼 ..솔직히 재미는없다..평점 조정	0
6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던ㅋ	
5403919	막 걸음마 뗀 3세부터 초등학교 1학년생인 8살용영화.ㅋㅋㅋ...별반개도 아까움.	0
7797314	원작의 긴장감을 제대로 살려내지못했다.	0
9443947	별 반개도 아깝다 육나온다 이응경 길용우 연기생활이몇년인지..정말 발로해도 그것	
7156791	액션이 없는데도 재미 있는몇안되는 영화	1
5912145	왜케 평점이 낮은건데? 꽤 불만한데.. 헐리우드식 화려함에만 너무 길들여져 있나?	
9008700	강인피니트가짱이다.진짜짱이다♥	1
10217543	볼때마다 눈물나서 죽겠다90년대의 향수자극!!허진호는 감성절제멜로의 달인이다.	
5957425	울면서 손들고 횡단보도 건널때 뛰쳐나올뻔 이범수 연기 드럽게못해	0
8628627	담백하고 깔끔해서 좋다. 신문기사로만 보다 보면 자꾸 잊어버린다. 그들도 사람이었	
9864035	취향은 존중한다지만 진짜 내생에 극장에서 본 영화중 가장 노잼 노감동임 스토리도	
6852435	그냥 매번 긴장되고 재밋음ㅠㅠ	1
9143163	참 사람들 웃긴게 바스코가 이기면 락스코라고 까고바비가 이기면 아이돌이라고 깐	
4891476	굿바이 레닌 표절인것은 이해하는데 왜 뒤로 갈수록 재미없어지니	0
7465483	이건 정말 깨알 캐스팅과 질퍽하지않은 산뜻한 내용구성이 잘 버무려진 깨알일드!!♥	
3989148	약탈자를 위한 변명, 이라. 저놈들은 착한놈들 절대 아닌걸요.	1
4581211	나름 심오한 뜻도 있는듯. 그냥 학생이 선생과 놀아나는 영화는 절대 아님	1
2718894	보면서 웃지 않는 건 불가능하다	1
9705777	재미없다 지루하고. 같은 음식 영화인데도 바베큐의 만찬하고 넘 차이남....바베큐의	
471131	절대 평범한 영화가 아닌 수작이라는걸 말씀드립니다.	1

자, 그럼 본격적으로 시작해볼까요?

1. KoNLPy로 데이터 전처리
2. NLTK로 데이터 탐색
3. Bag-of-words(term existence)로 문서 표현하고 classify
4. Gensim으로 doc2vec으로 문서 표현하고 classify

1. Data preprocessing (feat. KoNLPy)

- 데이터 읽기

```
def read_data(filename):
    with open(filename, 'r') as f:
        data = [line.split('\t') for line in f.read().splitlines()]
        data = data[1:] # header 제외
    return data

train_data = read_data('ratings_train.txt')
test_data = read_data('ratings_test.txt')
```

NOTE: `pandas.read_csv()`를 사용해서 데이터를 읽을 경우에는 `quoting parameter`를 `csv.QUOTE_NONE (3)`로 명시해주지 않으면 파싱이 제대로 안 됨 (ex: 리뷰 앞부분에 쌍따옴표가 있는 경우)

1. Data preprocessing (feat. KoNLPy)

- 데이터가 제대로 읽혔는지 확인

```
# row, column의 수가 제대로 읽혔는지 확인
print(len(train_data))      # nrows: 150000
print(len(train_data[0]))    # ncols: 3

print(len(test_data))       # nrows: 50000
print(len(test_data[0]))    # ncols: 3
```

1. Data preprocessing (feat. KoNLPy)

- 형태소로 토크나이징

```
from konlpy.tag import Twitter
pos_tagger = Twitter()

def tokenize(doc):
    # norm, stem은 optional
    return ['/'.join(t) for t in pos_tagger.pos(doc, norm=True, stem=True)]

train_docs = [(tokenize(row[1]), row[2]) for row in train_data]
test_docs = [(tokenize(row[1]), row[2]) for row in test_data]

# 잘 들어갔는지 확인
from pprint import pprint
pprint(train_docs[0])
# => [('의/Exclamation',
#       '더/Verb',
#       '빙/Noun',
#       '..../Punctuation',
#       '진짜/Noun',
#       '짜증/Noun',
#       '나다/Verb',
#       '목소리/Noun'],
#       '0')]
```

NOTE: 스크립트를 실행할 때마다 형태소 분석을 하지 않아도 되도록 분석 결과를 파일로 저장하고 가는 것도 좋습니다.

1. Data preprocessing (feat. KoNLPy)

- "그런데 형태소로 꼭 나눠야하나요?"
 - 데이터가 정말 충분하다면, 어절 단위로도 분석 가능하겠지만 우리는 해당 x
 - 한편 최근 히트를 친 시인뉴럴은 음절 단위의 분석 *
 - 토큰의 단위는 선택의 문제
 - "꼭 품사(POS) 태그를 부착해야하나요?"
 - 그것도 선택의 문제
 - 하지만 품사를 태깅해두면 동음이의어를 구분할 수 있다는 장점이 있음
 - ex: '은/Josa' vs '은/Noun'

```
| 25s 9326/345930 (epoch 8.089), train_loss = 1.2151  
| 34s 9327/345930 (epoch 8.089), train_loss = 1.2588  
| 22s  
time/ 소리를 넣는 날  
JAC 이상진  
바다 바람 부는 바람이  
우리집에는 나무를 물러보며  
작은 놀라운 풍경을  
너무 많은 사람 안에 나와  
마음에서 서 있는  
사람을 보면  
당신의 사랑은 아니  
어떻게 되었지요  
여전 눈물이다.  
... 한숨 속으로 다시금 앉았다  
picture 0 내 마음을 이겨내는 이유는  
나를 나는 알게 되었다  
사랑하는 이들이 있어서 니다  
나는 아무도 없습니다  
다른 이유를 갖는 기반의 물리적 이들이 있다.  
그리고 그들이 저에게 보내온 드론들에 가
```

* <https://github.com/karpathy/char-rnn>

2. Data exploration (feat. NLTK)

- 우리의 코퍼스가 어떤 특징들을 가지고 있는지 살펴봅시다!
- Training data의 token 모으기

```
tokens = [t for d in train_docs for t in d[0]]  
print(len(tokens))  
# => 2194536
```

2. Data exploration (feat. NLTK)

- nltk.Text()를 써보자! (Exploration이 편함)
 - 원래 단일 document를 분석할 때는 훨씬 풍부한 기능들을 사용할 수 있지만, 여기서는 그 중 몇 가지만 살펴봅시다.

```
import nltk
text = nltk.Text(tokens, name='NMSC')
print(text)
# => <Text: NMSC>
```

2. Data exploration (feat. NLTK)

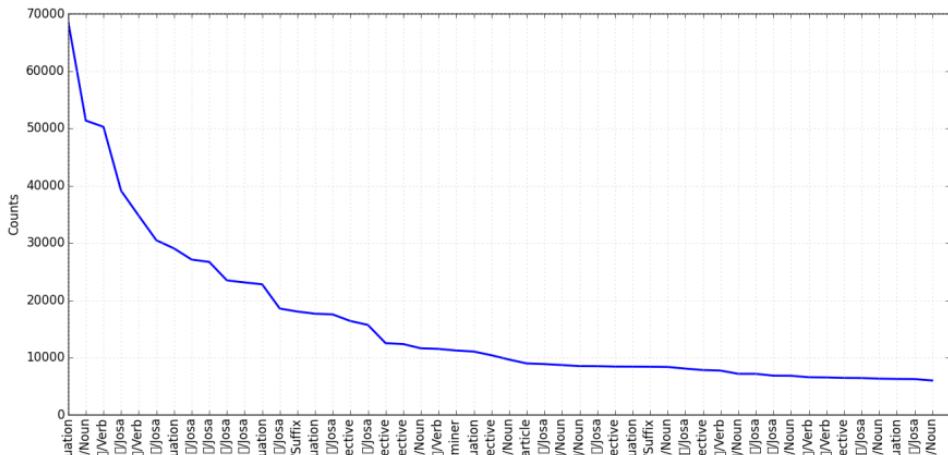
- Count tokens

```
print(len(text.tokens))      # returns number of tokens
# => 2194536
print(len(set(text.tokens))) # returns number of unique tokens
# => 48765
pprint(text.vocab().most_common(10)) # returns frequency distribution
# => [('./Punctuation', 68630),
#     ('영화/Noun', 51365),
#     ('하다/Verb', 50281),
#     ('으/Josa', 39123),
#     ('보다/Verb', 34764),
#     ('으/Josa', 30480),
#     ('./Punctuation', 29055),
#     ('으/Josa', 27108),
#     ('가/Josa', 26696),
#     ('을/Josa', 23481)]
```

2. Data exploration (feat. NLTK)

- Plot tokens by frequency

```
text.plot(50) # Plot sorted frequency of top 50 tokens
```



어라 글자 깨짐;;

2. Data exploration (feat. NLTK)

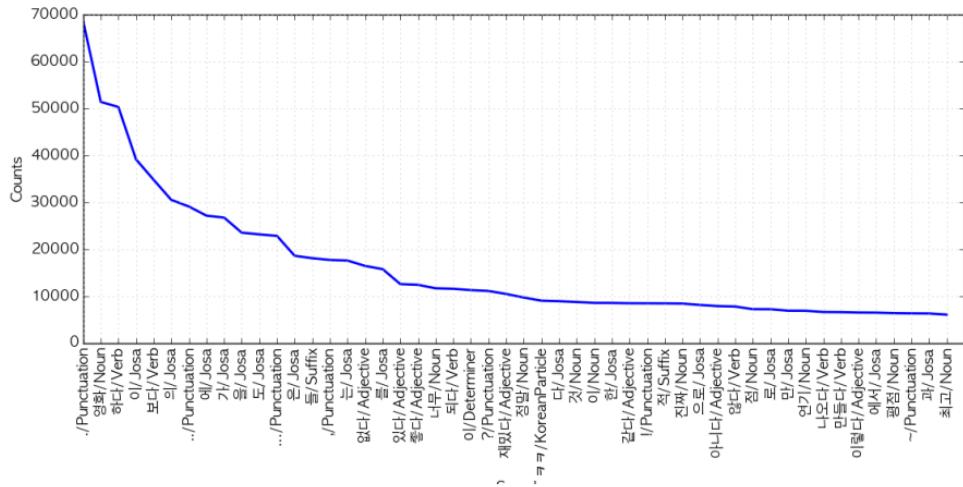
- 이건 폰트 문제!

Troubleshooting: For those who see rectangles instead of letters in the saved plot file, include the following configurations before drawing the plot:

- Some example fonts:
 - Mac OS: /Library/Fonts/AppleGothic.ttf

```
from matplotlib import font_manager, rc
font_fname = 'c:/windows/fonts/gulim.ttc'      # A font of your choice
font_name = font_manager.FontProperties(fname=font_fname).get_name()
rc('font', family=font_name)
```

2. Data exploration (feat. NLTK)



앞 장표 코드 실행 후. 굿.

일단 그림을 그리고 보니 영화에 대한 코퍼스라는건 확실히 알겠다.

2. Data exploration (feat. NLTK)

- Collocations (연어): 인접하게 빈번하게 등장하는 단어 (ex: "text" + "mining")

```
text.collocations() # 시간이 좀 오래 걸릴수도  
# => 이/Determiner 것/Noun; 적/Suffix 인/Josa; 이/Determiner 거/Noun; 안/Noun  
# 되다/Verb; 것/Noun 은/Josa; 10/Number 점/Noun; 배우/Noun 들/Suffix; 수/Noun  
# 있다/Adjective; 이/Noun 게/Josa; 내/Noun 가/Josa; 최고/Noun 의/Josa; 네/Suffix  
# 요/Josa; 이/Noun 영화/Noun; 끝/Noun 까지/Josa; 들/Suffix 이/Josa; 봐/Noun  
# 도/Josa; 때문/Noun 에/Josa; 적/Suffix 으로/Josa; 사람/Noun 들/Suffix; 영화/Noun  
# 를/Josa
```

NOTE: `nltk.Text()`에 대한 더 자세한 정보는, [source code](#)나 [API](#)에서 볼 수 있다. 영문과 한글의 동시 적용 예는 [이 링크](#) 확인.

3. Sentiment classification with term-existance*

- 여기서는 간단하게 term이 문서에 존재하는지의 유무에 따라 분류를 해보자.

```
# 여기서는 최빈도 단어 2000개를 피쳐로 사용  
# WARNING: 쉬운 이해를 위한 코드이며 time/memory efficient하지 않습니다
```

```
selected_words = [f[0] for f in text.vocab().most_common(2000)]
```

```
def term_exists(doc):  
    return {'exists({})'.format(word): (word in set(doc)) for word in selected_words}
```

```
# 시간 단축을 위한 품수로 training corpus의 일부만 사용할 수 있음  
train_docs = train_docs[:10000]
```

```
train_xy = [(term_exists(d), c) for d, c in train_docs]  
test_xy = [(term_exists(d), c) for d, c in test_docs]
```

Try this:

- 최빈도 단어들은 의미가 없을 수 있다. 최상위 50개를 제거하고 분석하면?
- TF, TF-IDF의 성능은?
- 코드 최적화(sparse matrix) or scikit-learn의 TfidfVectorizer()를 이용!

3. Sentiment classification with term-existance

- 이제 마지막 단계. 분류!
- 다양한 분류기를 사용해보면 재밌다.
- NLTK에서 제공하는 것만도 여러 개:
 - Naive Bayes Classifiers
 - Decision Tree Classifiers
 - Maximum Entropy Classifiers
 - 등등.
- scikit-learn 등 다른 패키지의 분류기도 얼마든지 사용 가능함.
- 여기서는 NLTK의 Naive Bayes Classifier 하나만 사용해 보자!

3. Sentiment classification with term-existance

- Naive Bayes classifier 적용

```
# 초간단함
classifier = nltk.NaiveBayesClassifier.train(train_xy)
print(nltk.classify.accuracy(classifier, test_xy))
# => 0.80418
classifier.show_most_informative_features(10)
# => Most Informative Features
# exists(수작/Noun) = True      1 : 0    =    38.0 : 1.0
# exists(최악/Noun) = True      0 : 1    =    30.1 : 1.0
# exists(♥/Foreign) = True      1 : 0    =    24.5 : 1.0
# exists(노잼/Noun) = True      0 : 1    =    22.1 : 1.0
# exists(낭비)/Noun) = True     0 : 1    =    19.5 : 1.0
# exists(쓰레기)/Noun) = True   0 : 1    =    19.4 : 1.0
# exists(여운)/Noun) = True     1 : 0    =    18.9 : 1.0
# exists(발연기)/Noun) = True   0 : 1    =    16.9 : 1.0
# exists(굿)/Noun) = True       1 : 0    =    16.9 : 1.0
# exists(최고다)/Noun) = True   1 : 0    =    15.9 : 1.0
```

- 긍정/부정 중 아무거나 찍어도 맞출 확률은 반반이니까 baseline accuracy는 0.50
- 그에 비해 우리는 리뷰 1만개만 쓴 경우에도 **accuracy가 0.80이** 나왔다. 나름 성공적?

4. Sentiment classification with doc2vec (feat. Gensim)

- 이제 앞서 살펴본 doc2vec으로 리뷰를 긍정/부정으로 분류해봅시다.
- Gensim을 사용할거에요.*

The screenshot shows the official Gensim website. At the top, there's a navigation bar with links for Home, Tutorials, Install, Support, API, and About. To the right of the navigation is a green 'Download' button with the subtext 'Latest version from the Python Package Index'. Below that is another button labeled 'Direct install with: easy_install -U gensim'. The main content area features the Gensim logo and the tagline 'topic modelling for humans'. On the left, there's a code snippet demonstrating how to use the library:

```
>>> from gensim import corpora, models, similarities
>>> # Load corpus iterator from a Matrix Market file on disk.
>>> corpus = corpora.MmCorpus('/path/to/corpus.mm')
>>>
>>> # Initialize Latent Semantic Indexing with 200 dimensions.
>>> lsi = models.LsiModel(corpus, num_topics=200)
>>>
>>> # Convert another corpus to the Latent space and index it.
>>> index = similarities.MatrixSimilarity(lsi[another_corpus])
>>>
>>> # Compute similarity of a query vs. indexed documents
>>> sim = index[query]
```

To the right of the code, the text 'Gensim is a FREE Python library' is displayed. Below this, three features are listed with checkmarks:

- Scalable statistical semantics
- Analyze plain-text documents for semantic structure
- Retrieve semantically similar documents

만든이]: Radim Rehureck

* 여기서는 Gensim 0.12.0을 사용하였습니다. 참고로 업데이트가 활발한 라이브러리가 버전업이 빨라요!

4. Sentiment classification with doc2vec (feat. Gensim)



Radim Řehůřek
@RadimRehurek



Following

Setting up in our new home, at Geoje Island
(거제도), Korea. Exciting :)



아니 메인테이너가 거제도에!!!
Gensim에 대해 궁금한게 있으면 그 곳으로...

4. Sentiment classification with doc2vec (feat. Gensim)

- doc2vec는 약간 더 복잡합니다.
- doc2vec이 요구하는 형태로 데이터를 바꿔줘야하기 때문
 - from gensim.models.doc2vec import TaggedDocument를 이 용해도 OK

```
from collections import namedtuple  
  
TaggedDocument = namedtuple('TaggedDocument', 'words tags')
```

```
# 여기서는 15만개 training documents 전부 사용함  
tagged_train_docs = [TaggedDocument(d, [c]) for d, c in train_docs]  
tagged_test_docs = [TaggedDocument(d, [c]) for d, c in test_docs]
```

4. Sentiment classification with doc2vec (feat. Gensim)

```
from gensim.models import doc2vec

# 사전 구축
doc_vectorizer = doc2vec.Doc2Vec(size=300, alpha=0.025, min_alpha=0.025, seed=1234)
doc_vectorizer.build_vocab(tagged_train_docs)

# Train document vectors!
for epoch in range(10):
    doc_vectorizer.train(tagged_train_docs)
    doc_vectorizer.alpha -= 0.002 # decrease the learning rate
    doc_vectorizer.min_alpha = doc_vectorizer.alpha # fix the learning rate, no decay

# To save
# doc_vectorizer.save('doc2vec.model')
```

Try this:

- Vector size, epoch 수 등 각종 파라미터 바꿔보기

4. Sentiment classification with doc2vec (feat. Gensim)

- 제대로 학습 됐는지 확인해보자! (1/3)

```
pprint(doc_vectorizer.most_similar('공포/Noun'))
#=> [('서스펜스/Noun', 0.5669919848442078),
#     ('미스터리/Noun', 0.5522832274436951),
#     ('스릴러/Noun', 0.5021427869796753),
#     ('장르/Noun', 0.5000861287117004),
#     ('판타지/Noun', 0.4368450343608856),
#     ('무계/Noun', 0.42848479747772217),
#     ('호러/Noun', 0.42714330554008484),
#     ('환타지/Noun', 0.41590073704719543),
#     ('멜로/Noun', 0.41056352853775024),
#     ('공포영화/Noun', 0.4052993059158325)]
```

- 오... 괜찮다.

4. Sentiment classification with doc2vec (feat. Gensim)

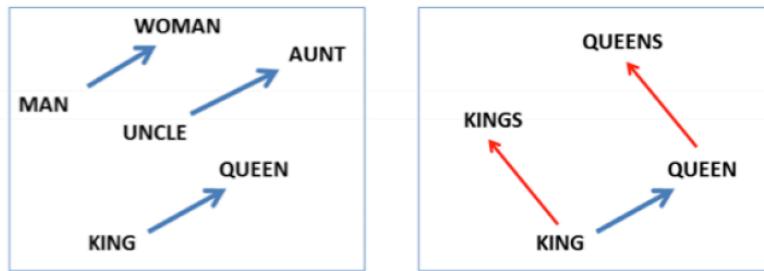
- 제대로 학습 됐는지 확인해보자! (2/3)

```
pprint(doc_vectorizer.most_similar('ㅋㅋ/KoreanParticle'))  
#=> [('ㅋㅋ/KoreanParticle', 0.5768033862113953),  
#      ('ㅋ/KoreanParticle', 0.4822016954421997),  
#      ('!!!!/Punctuation', 0.4395076632499695),  
#      ('!!!/Punctuation', 0.4077949523925781),  
#      ('!!/Punctuation', 0.4026390314102173),  
#      ('~/Punctuation', 0.40038347244262695),  
#      ('~~/Punctuation', 0.39946430921554565),  
#      ('!/Punctuation', 0.3899948000907898),  
#      ('^/^/Punctuation', 0.3852730989456177),  
#      ('~^/^/Punctuation', 0.3797937035560608)]
```

- 굿! 결과가 재밌네요.

4. Sentiment classification with doc2vec (feat. Gensim)

- 엇, 그리고보니 그 유명한 $v(\text{KING}) - v(\text{MAN}) + v(\text{WOMAN}) = v(\text{QUEEN})$ 의 관계도 성립할까?



(Mikolov et al., NAACL HLT, 2013)

4. Sentiment classification with doc2vec (feat. Gensim)

- 제대로 학습 됐는지 확인해보자! (3/3)

```
pprint(doc_vectorizer.most_similar(positive=['여자/Noun', '왕/Noun'], negative=['남자/Noun']))  
#=> [('악당/Noun', 0.32974398136138916),  
#      ('곽지민/Noun', 0.305545836687088),  
#      ('심영/Noun', 0.2899821400642395),  
#      ('오빠/Noun', 0.2856029272079468),  
#      ('전작/Noun', 0.2840743064880371),  
#      ('눈썹/Noun', 0.28247544169425964),  
#      ('광팬/Noun', 0.2795347571372986),  
#      ('지능/Noun', 0.2794691324234009),  
#      ('박보영/Noun', 0.27567577362060547),  
#      ('강예원/Noun', 0.2734225392341614)]
```

- 음...? 왜죠?

4. Sentiment classification with doc2vec (feat. Gensim)

```
text.concordance('왕/Noun', lines=10)
```

```
# => Displaying 10 of 145 matches:
```

```
# Josa 로맨스/Noun 냐/Josa ./Punctuation 왕/Noun 짜증/Noun ...../Punctuation 아주/Noun 전  
# /Noun 함/Noun ..//Punctuation 결말/Noun 왕/Noun 실망/Noun 임/Noun 전작/Noun 에/Josa 비/No  
# nction 얼굴/Noun 만/Josa 예쁘다/Adjective 왕/Noun 되다/Verb 맞다/Verb 드라마/Noun 라도/Jos  
# /Noun 스릴러/Noun 임/Noun ??/Punctuation 왕/Noun 실망/Noun ./Punctuation 연기/Noun 대본/N  
# b 금/Noun 사인방/Noun ㅠㅠ/KoreanParticle 왕/Noun 잽/Noun 없다/Adjective ./Punctuation 정  
# osa 서유기/Noun 보다/Josa 희극/Noun 지/Josa 왕/Noun 이/Josa 더/Noun 죄고/Noun 라/Josa 생  
# 접/Noun 한/Josa 걸작/Noun ./Punctuation 왕/Noun ./Punctuation 너무/Noun 감동/Noun 적/Suf  
# Josa 온/Noun 거/Noun 처럼/Noun 제나라/Noun 왕/Noun 과/Josa 군사/Noun 들/Suffix 을/Josa 속  
# 다/Verb ./Punctuation 기대하다/Adjective 왕/Noun 지루/Noun ..//Punctuation 제니퍼/Noun 틸리  
# tive 움/Noun 짜증/Noun .../Punctuation 왕/Noun 짜증/Noun ..//Punctuation 사람/Noun 마다/J
```

- 아, 그랬군요;;
- 사실, 여기서 또 재밌는 문제가 발생합니다. 어떻게 다의어를 구분할 수 있는가? *

4. Sentiment classification with doc2vec (feat. Gensim)

이제 마지막 분류 단계. 분류를 위한 피쳐들을 마련하자.

```
train_x = [doc_vectorizer.infer_vector(doc.words) for doc in tagged_train_docs]
train_y = [doc.tags[0] for doc in tagged_train_docs]
len(train_x)    # 사실 이 때문에 앞의 term existance라는 공평한 비교는 아닐 수 있다
# => 150000
len(train_x[0])
# => 300
test_x = [doc_vectorizer.infer_vector(doc.words) for doc in tagged_test_docs]
test_y = [doc.tags[0] for doc in tagged_test_docs]
len(test_x)
# => 50000
len(test_x[0])
# => 300
```

4. Sentiment classification with doc2vec (feat. Gensim)

- 이번에는 scikit-learn에서 LogisticRegression() 클래스를 빌려 오자.

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=1234)
classifier.fit(train_x, train_y)
classifier.score(test_x, test_y)
# => 0.7824600000000004
```

- 흠! Accuracy가 0.78이군요.
- Term existance 전략에 비해 accuracy가 2% 정도 감소했지만, 나쁘지 않네요.
- 기억해야 할 것은, term existance도, doc2vec도 시도해볼만한 다양한 변형들이 많다는 것!

마무리

- 텍스트의 다양한 표현법에 대해 살펴보았다.
- KoNLPy로 데이터를 전처리하고, NLTK로 데이터를 탐색하고, Gensim으로 문서 벡터를 구해봤다.
- Term existance와 document vector로 센티멘트를 분류해봤다.
- 여기서는 영화 리뷰에 대한 센티멘트 분석을 했지만 얼마든지 다른 task에도 apply 할 수 있음!
 - 국회 의안 통과/폐기 예측 *
 - 주가 상승/하락 예측
- 나는 어떤 태스크에 적용해볼 수 있을지 생각해보자!

* 박은정, 강필성, 조성준, 어떤 의안이 법률이 되는가: 데이터 불균형을 고려한 의안 결과 예측, SNUDM Technical Reports, Apr 14, 2015.

감사합니다 :D

<http://lucypark.kr>
@echojuliett

